`

Generating and Testing a Random Stream using dynamic S-box

Ambika H Shetty

National Institute of Technology Karnataka
Surathkal, Mangalore, India

Abstract - This paper discusses about a new algorithm in the area of cryptography. The algorithm is expected to find an extensive use in wireless telecom networks for end-to-end network security. A new stream cipher based on dynamic S-box is designed, implemented and tested in MATLAB. The new cipher thus generated is tested through 16 different randomness tests mentioned in one of the NIST special publication 800-22[2].For this implementation many references have been taken from the already existing Advanced Encryption Standard structure. This work was done as part of the fulfilment of Master in Technology at Manipal Institute of Technology, Manipal.

I. Introduction

We need security to feel safe in life. Whether it is about feeling or about the information security in computer network, security has always been one of the main concerns. One of the several available methods to secure the information in any network is cryptography. Stream ciphers and block ciphers are one of the classifications of cipher structures. In this paper we concentrate mainly on stream ciphers since our work and the findings are on stream ciphers.

Stream ciphers are essentially meant to be a pseudorandom generator. So this implementation work aims to generate a stream that is random in nature. To verify the randomness of the resulting stream we apply around 16 randomness tests on the stream. For that, we first implement those 16 tests in Matlab. And to check the correctness of the Matlab implementation of these 16 tests, we apply these tests first on one of the already existing stream cipher. And the reference stream cipher we take for this purpose is RC4 stream cipher.

II. RC4 As Reference Algorithm

There is a high need to improve the existing stream ciphers by introducing more non-linearity into it.Before doing this we intend to study any of the existing stream cipher algorithm so as to make a comparative study. And the algorithm that we choose for this purpose is the RC4 algorithm.

RC4 is a stream cipher designed in 1987 by Ron Rivest.It is a variable key-size stream cipher with byte-oriented operations. The algorithm is based on the use of a random permutation. It is the most widely used cipher. The algorithm is based on the use of a random permutation. Analysis shows that the period of the cipher is overwhelmingly likely to be greater than 10^{100}.Eight to sixteen machine operations are required per output byte, and the cipher can be expected to run very quickly in software[1] .

Taking all these factors into consideration we come to a conclusion that RC4 is the best choice to start this work. Accordingly we start with the implementation of RC4 algorithm in Matlab.

III. Statistical Test Suite for Validating Randomness

Next work is to validate the RC4 key stream as random. The intention behind this validation is not to test the RC4 algorithm instead to make sure the correctness in the implementation of those 16 tests that are to be run on RC4.

NIST issued a special publication (800-22) that discusses about aspects of selecting and testing random and pseudorandom number generators [2]. The generators suitable for use in cryptographic applications may need to meet stronger requirements than for other applications. NIST publication discusses about some

`

criteria for characterizing and selecting appropriate generators. With help of these criteria we can decide whether our new stream cipher generated is a perfect random generator or not.

Randomness is a probabilistic property. Various statistical tests can be applied to a sequence to attempt to compare and evaluate the sequence to a truly random sequence. The properties of a random sequence can be characterized and described in terms of probability. For each applied test, decision or conclusion is derived that accepts or rejects the sequence that was produced. The statistic value is a function of the data. Test statistic is used to calculate a P-value that summarizes the strength of the stream generated. For these tests; each P-value is the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested , given the kind of non-randomness assessed by the test. A P-value ≥ 0.01 would mean that the sequence would be considered to be random with a confidence of 99%.A P-value < 0.01 would mean that the sequence is non-random with a confidence of 99%[2].

IV. Random Number Generation Tests

The NIST Test suite is a statistical package consisting of 16 tests that were developed to test the randomness of binary sequences produced by either hardware or software based cryptographic random or pseudorandom number generators. The 16 tests are:

1. The frequency Test: Purpose is to determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence.

2. Frequency Test within a block: Purpose of this test is to determine whether the frequency of ones in an M-bit block is approximately M/2, as would be expected under an assumption of randomness.

3. Runs Test: Purpose is to determine whether the number of ones and zeros of various lengths is as expected for a random sequence.

4. Test for the Longest Run of ones in a block: Purpose is to determine whether the length of the longest run of ones that would be expected in a random sequence.

5. Binary Matrix Rank Test: Purpose is to check for linear dependence among fixed length substrings of the original sequence.

6. Discrete Fourier Transform (Spectral) Test: Purpose is to detect the periodic features in the tested sequence that would indicate a deviation from the assumption of randomness.

7. Non- overlapping Template Matching Test: Purpose is to detect generators that produce too many occurrences of a given non-periodic pattern. For this test an m-bit window is used to search for a specific m-bit pattern. If the pattern is not found, the window slides one bit position. If the pattern is found, the window is reset to the bit after the found pattern, and the search resumes.

8. Overlapping Template matching Test: The difference between this test and test number 7 is that when the pattern is found, the window slides only one bit before resuming the search.

9. Maurer's Universal Statistical Test: Purpose of this test is to detect whether or not the sequence can be significantly compressed without loss of information.

10. Lempel-Ziv Compression Test: Purpose is to determine how far the tested sequence can be compressed.

11. Linearity Complexity Test: Purpose of this test is to determine whether or not the sequence is complex enough to be considered random.

`

12. Serial Test: Purpose is to determine whether the number of occurrences of the 2m m-bit overlapping patterns is approximately the same as would be expected for a random sequence.

13. Approximate Entropy Test: Purpose is to compare the frequency of overlapping blocks of two consecutive lengths against the expected result for a random sequence.

14. Cumulative Sums Test: Purpose is to determine whether the cumulative sum of the partial sequences occurring in the tested sequence is too large or too small relative to the expected behavior of that cumulative sum for random sequences.

15. Random Excursions Test: Purpose is to determine if the number of visits to a particular state within a cycle deviates from what one would expect for a random sequence.

16. Random Excursions variant Test: Purpose is to detect deviations from the expected number of visits to various states in the random walk.

All the above tests are implemented in Matlab and we run them on RC4 key stream. We find that all the tests are successfully run on RC4 and give a P- value ≥ 0.01. We can now proceed with the implementation of new stream cipher based on dynamic S-box.

V. Random Number Generation Tests

For new stream cipher generation we make use of the AES Key expansion method [1]. But here we use a dynamic S-box which is dependent on input key value. Then by using this dynamic S-box we generate key stream by AES Key expansion method.

To begin with stream cipher generation, we generate an S-box depending on the input key value. We input a 16 byte key value. The binary equivalent of this 16 byte key is stored in an array. We find a non-singular matrix by selecting first 8*8 = 64 bits of the array. If the determinant of the 8 by 8 matrix formed by these 64 bits is zero then we take next 64 bits (done by one bit left shift of the binary values in the key array).Once we find a non-singular matrix from the input key, we store it in another temporary array so that it can be used later for S-box generation. We keep a dummy key which will be useful, incase if the first key fails to give a non-singular matrix.

Now we proceed with the generation of dynamic S-box by following the steps mentioned for AES algorithm. But here modulo polynomial need not be constant always. Selection of irreducible modulo polynomial is also left to the user. Also while doing affine transformation ax + b = c, the value of 'c' is also a user's choice. The matrix 'a' in the affine transformation is the one obtained by the input key [1].

Thus the resulting S-box cannot be predicted just by looking at the input key and for different input keys we are able to generate different S-box providing non-linearity to the cipher structure. We use this dynamic S-box providing non-linearity to the cipher structure. We use this dynamic S-box in AES Key expansion method to produce the key stream [4].

Finally to validate the randomness of the resulting key stream we apply the NIST tests on key stream. Figures 1 and 2 give a flowchart of the implementation work carried out. Key Stream shown in figure 2 is the generated random stream upon which FIPS mentioned 16 tests were successfully tested.

`

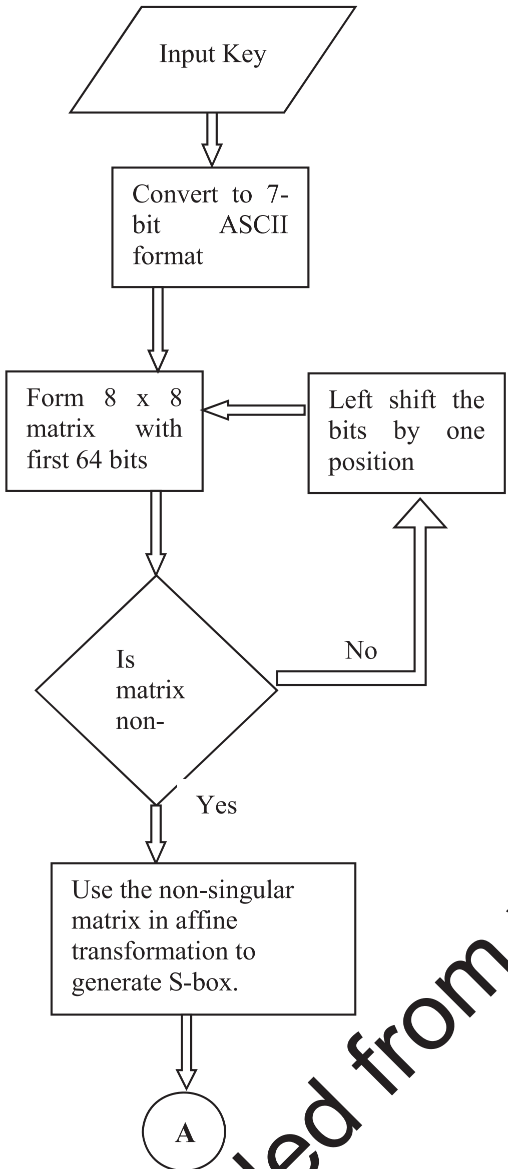

Figure 1. S-box generation

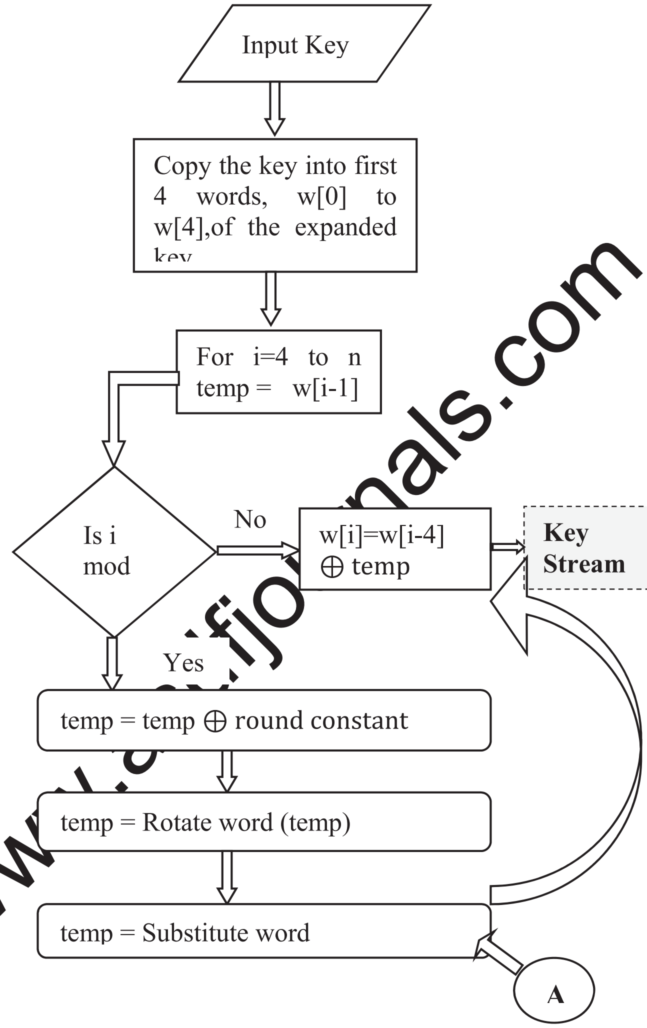

Figure 2. Key generation

VI. Preliminary Result Analysis

As explained earlier, before we generate and test our new stream cipher we should be sure about the working of the 16 randomness tests we have implemented. For this purpose we take RC4 as the reference algorithm because RC4 already known as a perfect random stream generator [4]. If all the 16 randomness tests successfully run on RC4 and gives the desired result i.e., a P-value ≥ 0.01, then we can be sure about working / implementation of 16 randomness tests.

When we tested RC4 key stream with the 16 randomness tests we obtained a P-value ≥ 0.01 for all 16 tests. With this preliminary result it was easy to go ahead with our new stream cipher generation without any confusion about the correctness in the implementation of 16 randomness tests.

VI. Results

We implemented all 16 tests in MATLAB and then these tests were run on the RC4 generated key stream. We first generate 256 bytes of key stream. The generated key stream is tested for its randomness. Table 4.1 gives the result obtained when the tests were run on different lengths of key stream. As discussed in earlier chapters the method of validating the randomness of key stream is by checking the P-value. If the obtained P-value is less than 0.01 then the key stream is declared as non-random. If the P-value is greater than or equal to 0.01 then the key stream is declared as a valid random key stream.

One fact to be observed here is that RC4 is already known for the randomness in its resulting key stream. Therefore while testing RC4 we have not considered more than 3 to 4 scenarios. If by running at least 3 scenarios we can obtain a P-value of greater than or equal to 0.01 that itself is a valid result to proceed for the next step. The NIST Test suite however mentions that more the number of scenarios passed, better is the resulting algorithm. While testing our new stream cipher therefore we consider many different scenarios by taking different input keys. But only selected 5 scenarios are highlighted in table 1.

All the columns shown in table 1 are tested for a key stream length of 8800 bytes. We test the key stream for different values of input key. There are two different input keys used in our implementation. One is to generate the dynamic S-box and the other to use for the key expansion routine. The key used for key expansion routine is changed and for all the changed values of key we get different key streams. All the key streams are tested for more than 15 scenarios and were observed to pass the tests successfully. In table 1 we have highlighted only few test scenarios. Lempel-Ziv test result has not been tabulated for the reason that few of the parameters (mean and variance) required to calculate P-value in this test are not provided in the publication 800-22.So we neglect this test.

VI. How Better is Our New Stream Cipher Algorithm?

Now that we have seen and analyzed the test results for both RC4 and our new stream cipher based on dynamic S-box we should be able to differentiate the level of security provided by both the algorithms.

We see that both RC4 as well as new stream cipher pass NIST tests. Now the improvement or the increase in amount of security provided by the new stream cipher lies solely in the fact that the S-box used for the implementation is highly unpredictable. None of the existing stream ciphers are making use of S-box for the key stream generation which itself is a big plus point in our new stream cipher generation.

No doubt RC4 has been the most widely used stream cipher so far because of its simplicity of implementation; we see that there are still few loopholes in RC4 as mentioned in literature survey. So our new stream cipher would probably be the better choice for more advanced applications.

Table 1 P-values for new key stream

	All bits of input key = '0'	All bits of input key = '1'	Random key	All except last one bit of key='1'	All except last one bit of key='0'
Test #1	0.1918	0.5922	0.0775	0.2362	0.3981

`

Test #2	0.9670	1.0000	0.9980	1.0000	0.0202
Test #3	0.6445	0.6300	0.0991	0.9801	0.7363
Test #4	0.9544	0.4862	0.6497	0.1187	0.0104
Test #5	0.5936	0.2101	0.5738	0.5868	0.6454
Test #6	0.5577	0.5577	0.2554	0.2561	0.5510
Test #7	0.0288	0.0288	0.1394	0.1394	0.0288
Test #8	0.1252	0.1257	0.1296	0.1255	0.1251
Test #9	0.0646	0.0127	0.0322	0.0908	0.0905
Test #11	0.5119	0.1927	0.7955	0.3084	0.5984
Test #12	0.6203	0.1260	0.5883	0.7211	0.3745
Test #13	0.5119	0.7719	0.6365	0.8334	0.9567
Test #14	0.9567	0.8344	0.8334	0.8145	0.7969
Test #15	0.8638	0.5922	0.8145	0.7680	0.8840
Test #16	0.4924	0.7628	0.0775	0.2362	0.3981

VI. Conclusion

Now that we have seen and analyzed the test results for both RC4 and our new stream cipher based on dynamic S-box we should be able to differentiate the level of security provided by both the algorithms.

The work concentrates only on the development of a new stream cipher based on AES dynamic S-box. While doing this we have given much attention to the randomness of the generated stream rather than how fast our algorithm runs. Hence performance analysis of the stream cipher algorithm that we have developed is still pending and to be taken as the part of future work. For checking the performance of the algorithm through software implementation we can go for a high level language like C/C++ and a better operating system like Linux of higher versions. We can also go for checking our stream cipher performance pertaining to a specific application. For this we can select any of the existing communication system where there is high requirement for encryption and decryption of data. Any security driven DSP application would be suitable choice for this.

Acknowledgment

Author would like to thank Dr T G S Chandrasekharappa, professor at Manipal Institute of technology, Manipal for his immense guidance and help throughout this project work.

References

1. William Stallings, "Cryptography and Network security" 4/e. Pearson, Prentice Hall.
2. NIST Special Publication 800-22,"A statistical test suite for Random and Pseudorandom number generators for cryptographic applications", May 15,2001.
3. S. Mister and S.E. Tavares," Cryptanalysis of RC4-lik ciphers", Pages 132 to 134,1999.
4. Krishnamurthy G N,V Ramaswamy,"Making AES Stronger: AES with key dependent S-box", Pages 388 to 394,Volume 8.No 9 September 2008.